Homework 2 Report Zhengxian Lin

Part 1

There is one file, which include three functions. Main() is for running this MDP; getMDPSATR(filename) is for get data from file, and distribute them to four variable, S, A, T, A; evaluationFun(s,h,satr) is dynamic program algorithm to calculate the value function and for k step-to-go and policy for k steps-to-go. I use a sample test case, the test case on the pdf, and two test cases (show in part 3) in the professor provided to test it. Finally, I create a test case (including in file summitted, show at part 2), which have real world meaning with 20 states to test it.

The sample test case is:

3 2

0.00 0.99 0.01
0.00 1.00 0.00
0.00 0.00 1.00

1.00 0.00 0.00
0.00 0.70 0.30
0.20 0.00 0.80

0.00 0.00
-1.00 -1.00
1.00 1.00

In this case, it is good to be state 2, because it can get reward forever. When the H is small, and the agent at the state 0, it cannot risk getting negative reward to get to state 2. However, if H is bigger than 6, it should do that.

Small test case, H = 10:

Value Function (col is horizon (10 ~ 1), row is states (0 ~ 2)):

2.6663	1.7805	0.9435	0.1765	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
3.5217	2.6024	1.7177	0.8824	0.1177	-0.5462	-1.0660	-1.3800	-1.4000	-1.0000
10.0000	9.0000	8.0000	7.0000	6.0000	5.0000	4.0000	3.0000	2.0000	1.0000

Policy (col is horizon (10 ~ 1), row is states (0 ~ 2)):

0	0	0	0	1	1	1	1	1	0
1	1	1	1	1	1	1	1	1	0
0	0	0	0	0	0	0	0	0	0

The result of test case on the pdf:

Value Function (col is horizon (10 ~ 1), row is states (0 ~ 2)):

-3.4600	-3.3021	-3.1441	-2.9862	-2.8289	-2.6748	-2.5260	-2.3500	-1.9500	-1.0000
-2.4073	-2.2494	-2.0916	-1.9336	-1.7755	-1.6174	-1.4620	-1.3200	-1.2000	-1.0000
-1.3547	-1.1968	-1.0389	-0.8810	-0.7231	-0.5650	-0.4063	-0.2475	-0.1000	0.0000

Policy Function (col is horizon (10 ~ 1), row is states (0 ~ 2)):

0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 0

Part 2

The TA told us that, it is better to come out a test case that have real world meaning. So, I come out a situation which can be test case for my algorithm. It is that you have 10 terms to take courses at a weird University.

State:

There are 20 levels (20 states) in this university, you only can take one course of your level at one term, e.g. if your level is 11, you only can take 11 level's courses. Cannot take the course you have taken. For each level, the number of course is infinite.

Actions:

1. Taking course: There is possibility to upgrade or keep level or degrade level after you take class. The possibility of getting higher grade is decreased when the level is higher.
2. Do nothing: keep your level and spend one term time.

Transition:

This university said that you can take course, but there are some rules:
If you got A, you upgrade your level, like 10 - 11
if you got B, you keep your level. like 10 - 10
if you got C you degrade one level, like 10 - 9
if you got D you degrade two level like 10 - 8;
if you got F you start from level 1, like 10 – 1
if you degrade level and it under level 1, you start from level one, like getting grade D on level 2, you start from level 1.

Reward:

1. Taking course:

Get -1 reward at lower level course (level1 – level 6) because the time and tuition are more important than the credit of low level.
Get 0 reward at middle level course (level 7 – level 12) because the time and tuition are equals to the credit of middle level.
Get 1 reward at high level course (level12 – level 20) because the credit of high level is more important than the time and tuition.

2. Staying on same level:

Staying on lower level lose more credit. Because time is money. For example, staying on level 1 get -1 rewards per term, staying on level 2 get -0.9 rewards per term.

The data is included on the file I summitted (20testcase.txt)

Result:

Value Function (col is horizon (10 ~ 1), row is states (0 ~ 19)):

-7.7864 -7.1090 -6.4313 -5.7527 -5.0723 -4.3838 -3.6070 -2.8010 -1.9010 -1.0000

-7.4606	-6.7832	-6.1057	-5.4281	-4.7494	-4.0692	-3.3816	-2.6050	-1.8000	-0.9000
-7.1281	-6.4507	-5.7733	-5.0958	-4.4182	-3.7397	-3.0605	-2.3750	-1.6000	-0.8000
-7.0000	-6.3000	-5.6000	-4.9000	-4.2000	-3.5000	-2.8000	-2.1000	-1.4000	-0.7000
-6.0000	-5.4000	-4.8000	-4.2000	-3.6000	-3.0000	-2.4000	-1.8000	-1.2000	-0.6000
-5.0000	-4.5000	-4.0000	-3.5000	-3.0000	-2.5000	-2.0000	-1.5000	-1.0000	-0.5000
-2.7420	-2.3901	-2.0371	-1.6822	-1.3231	-0.9648	-0.6256	-0.3409	-0.1300	0.0000
-2.2527	-1.9719	-1.6906	-1.4084	-1.1245	-0.8353	-0.5532	-0.2897	-0.1000	0.0000
-1.7000	-1.5000	-1.3000	-1.1000	-0.9000	-0.7000	-0.5000	-0.3000	-0.1100	0.0000
-0.9000	-0.8000	-0.7000	-0.6000	-0.5000	-0.4000	-0.3000	-0.2000	-0.1000	0.0000
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
0.6940	0.6940	0.6940	0.6940	0.6940	0.6940	0.6940	0.5957	0.3800	0.0000
2.1198	2.1198	2.1198	2.1198	2.1198	2.1198	2.0731	1.9110	1.5400	1.0000
2.3908	2.3908	2.3908	2.3908	2.3908	2.3908	2.3027	2.0636	1.6800	1.0000
2.5768	2.5768	2.5768	2.5768	2.5768	2.5768	2.4510	2.1521	1.6400	1.0000
3.3206	3.3206	3.3206	3.3206	3.2953	3.2104	3.0194	2.6342	1.9700	1.0000
1.5082	1.5082	1.5082	1.5082	1.5082	1.5082	1.5082	1.5082	1.4000	1.0000
1.7046	1.7046	1.7046	1.7046	1.7046	1.7046	1.7046	1.7046	1.6000	1.0000
1.2000	1.2000	1.2000	1.2000	1.2000	1.2000	1.2000	1.2000	1.2000	1.0000
1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000

Policy Function (col is horizon (10 ~ 1), row is states (0 ~ 19)):

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	0	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	0	0	0
1	1	1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	1	1	0
1	1	1	1	1	1	0	0	0	0
1	1	1	1	1	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0
1	1	1	1	1	1	1	0	0	0
1	1	1	1	1	1	1	0	0	0
1	1	1	1	1	1	1	1	0	0
1	1	1	1	1	1	1	1	0	0

Analysis:

The result is optimal solution. There can be divided to three-part of this solution, Low-level studying,

leve1 to level6 (0 – 5), Middle-level studying, level 7 to level 12 (6 - 11), and High-level studying, level 12 to level 20 (12 - 19).

For the first part, the action is changed in the [1, 8] and [2, 8], I call it “watershed”. Before the watershed, [row < 1 or (col < 8 and row < 3)], the punishment of staying on same level is too big. Because the future is long (col < 8), or it cannot be worse if keep studying (row < 1). After the watershed, [row >= 3 or (row > 1 and col >=8]. The punishment of staying on same level is smaller, and it is better to stay on same level than taking course and falling into lower level.

For the second part, the watershed is at level 10, because before level 10 (including 10), there are still punishments of staying in same level, and keep studying will not get negative reward, so, studying is a good choice. And after 10, there is not punishments of staying in same level, so, studying have possibility to fall into low level, so stay at same level is cleaver.

For third part, the watershed is at level16. This watershed is little bit tricky, because I change the possibility of falling to level 1 of level 16 to 0.02, which is much lower than its neighbor. That is why at the level 16, the agent decides to keep moving instead of staying on the same place. For the high level, it is cleaver to staying at same place although studying can get reward unless the future is short. It should keep away from falling into level 1, because the punishment is too heavy.

There is also one data that show the result is optimal. when the horizon == 1, and row > 5, which don’t get negative reward of studying, it always chooses studying. Because, studying can get reward and if it gets the grade less than B, falling into lower level, even level 1. It does not matter, because the future is end. So, just study.

Part 3

MDP1.txt test case, H = 10:

Value Function (col is horizon (10 ~ 1), row is states (0 ~ 9)):

3.0330	3.0000	3.0000	2.0330	2.0000	2.0000	1.0330	1.0000	1.0000	0.0000
2.9998	2.8944	2.0184	1.9998	1.8944	1.0184	0.9998	0.8944	0.0000	0.0000
2.8982	2.7992	2.0126	1.8982	1.7992	1.0126	0.8982	0.7987	0.0074	0.0000
2.9039	2.8944	2.0184	1.9039	1.8944	1.0184	0.9039	0.8944	0.0000	0.0000
4.0000	3.0330	3.0000	3.0000	2.0330	2.0000	2.0000	1.0330	1.0000	1.0000
2.8944	2.6524	1.9998	1.8944	1.6527	0.9998	0.8944	0.6595	0.0000	0.0000
2.9804	2.8783	2.6524	1.9804	1.8784	1.6527	0.9812	0.8806	0.6595	0.0000
2.8958	2.8524	2.0125	1.8958	1.8524	1.0125	0.8960	0.8522	0.0000	0.0000
3.0000	3.0000	2.0330	2.0000	2.0000	1.0330	1.0000	1.0000	0.0330	0.0000
3.0184	2.9039	2.8944	2.0184	1.9039	1.8944	1.0184	0.9039	0.8944	0.0000

Policy Function (col is horizon (10 ~ 1), row is states (0 ~ 9)):

3	3	3	3	3	3	3	3	3	0
1	3	3	1	3	3	1	3	0	0
1	2	2	1	2	2	1	2	2	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
2	0	2	2	0	2	2	0	0	0
1	1	1	1	1	1	1	1	1	0
2	2	2	2	2	2	2	2	0	0

1 1 1 1 1 1 1 1 3 0
3 0 3 3 0 3 3 0 3 0

Value Function (col is horizon (10 ~ 1), row is states (0 ~ 9)):

4.0569	4.0000	3.0569	3.0000	2.0569	2.0000	1.0569	1.0000	0.0569	0.0000
3.9952	3.9919	2.9952	2.9919	1.9952	1.9919	0.9950	0.9925	0.0000	0.0000
5.3085	4.6244	4.3081	3.6236	3.3062	2.6193	2.2970	1.5991	1.2527	0.4931
3.9927	3.0419	2.9928	2.0420	1.9929	1.0419	0.9936	0.0665	0.0006	0.0000
4.5715	4.0000	3.5715	3.0000	2.5715	2.0000	1.5715	1.0000	0.5715	0.0000
5.0000	4.0003	4.0000	3.0003	3.0000	2.0003	2.0000	1.0002	1.0000	0.0000
5.0003	5.0000	4.0003	4.0000	3.0003	3.0000	2.0002	2.0000	1.0000	1.0000
3.9648	3.0813	2.9648	2.0813	1.9648	1.0813	0.9657	0.0816	0.0775	0.0000
4.0000	3.5715	3.0000	2.5715	2.0000	1.5715	1.0000	0.5715	0.0112	0.0000
5.0000	4.0003	4.0000	3.0003	3.0000	2.0003	2.0000	1.0000	0.9999	0.0000

Policy Function (col is horizon (10 ~ 1), row is states (0 ~ 9)):

3	0	3	0	3	0	3	0	3	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	2	1	0
2	1	2	1	2	1	2	1	2	0
2	0	2	0	2	0	2	0	2	0
2	2	2	2	2	2	2	2	0	0
1	1	1	1	1	1	1	1	3	0
0	1	0	1	0	1	0	1	3	0
2	2	2	2	2	2	2	2	2	0